User's Manual

From Results to Decisions

Microsoft, Windows, Windows NT, Windows 2000, Windows XP, Visual Basic, Microsoft .NET, Visual C++, Visual C#, and ActiveX are either trademarks or registered trademarks of Microsoft Corporation.

All other nationally and internationally recognized trademarks and tradenames are hereby recognized.

Copyright © 2001-2008 by MVTec Software GmbH, München, Germany

Edition 1	September 2001	(ActivVisionTools 2.0)
Edition 2	November 2002	(ActivVisionTools 2.1)
Edition 3	January 2005	(ActivVisionTools 3.0)
Edition 4	February 2006	(ActivVisionTools 3.1)
Edition 5	May 2008	(ActivVisionTools 3.2)

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher.

More information about ActivVisionTools can be found at:

`http://www.activ-vision-tools.com`

How to Read This Manual

This manual explains how to evaluate results of other ActivVisionTools using ActivDecision. It describes the functionality of ActivDecision and its cooperation with other ActivVisionTools with Visual Basic examples. Before reading this manual, we recommend to read the manual Getting Started with ActivVisionTools, which introduces the basic concepts of ActivVisionTools, and the User's Manual for ActivView to learn how to load and display images.

For each example in this manual, there is a corresponding Visual Basic project; these projects can be found in the subdirectory `examples\manuals\activdecision` of the ActivVisionTools base directory you selected during the installation (default: `C:\Program Files\MVTec\ActivVisionTools`). Of course, you can also create your own Visual Basic projects from scratch. Note that the screenshots in the manual may differ slightly from the corresponding Visual Basic projects. To follow the examples actively, first install and configure ActivVisionTools as described in the manual Getting Started with ActivVisionTools.

We recommend to **create a private copy of the example projects** because by experimenting with the projects, you also change their *state*, which is then automatically stored in the so-called description files (extension `.dsc`) by ActivVisionTools. Of course, you can restore the state of a project by retrieving the corresponding description file from the CD.

Contents

1 About ActivDecision **1**

1.1 Introducing ActivDecision 2

1.2 The Sub-Tools of ActivDecision 4

2 Using ActivDecision **7**

2.1 Browsing Results 8

2.2 Formulating Conditions 10

2.3 Specifying Defaults Via a Context Menu 12

2.4 Specifying Defaults Via the Support Tool 14

2.5 Displaying Selected Results 16

3 Tips & Tricks **23**

3.1 Adapting the Display of Results in `AVTView` 24

3.2 Configuring the Two Execution Modes 26

3.3 Miscellaneous 28

3.4 Accessing Evaluation Results Via the Programming Interface 30

Chapter 1

About ActivDecision

This chapter will introduce you to the features and the basic concepts of the *decision tool* ActivDecision. It gives an overview about ActivDecision's *master tools* and *support tools*, which are described in more detail in chapter 2 on page 7.

1.1 Introducing ActivDecision 2
1.2 The Sub-Tools of ActivDecision 4

1.1 Introducing ActivDecision

In an ActivVisionTools application, the *vision tools*, e.g., ActivMeasure or ActivBarcode, calculate results (also called *features*) in form of numbers (e.g., positions or distances) or strings (e.g., a decoded bar code). Using ActivDecision, you can evaluate these results by formulating *conditions* they have to meet in order to be "okay". These evaluations are then treated as additional results, therefore you can output them together with the alphanumeric results using, e.g., ActivFile, ActivSerial, or ActivDigitalIO. Moreover, depending on these evaluations you can decide what is to be written to a log file or sent via the serial interface.

The Hierarchy of Results: Tool, ROI, and Object Features

Let's have a closer look at the "philosophy" of ActivVisionTools regarding alphanumeric results. Results stem from a two-step procedure: First, so-called *objects* are extracted from the image; in the case of ActivMeasure, e.g., objects correspond to edges or edge pairs, in the case of ActivBlobFinder to the blobs. Then, one or more *features* are calculated for each object. You can e.g. calculate the position of each edge but also its amplitude or the distance between edges; ActivFeatureCalc lets you calculate a large number of gray value and shape features for each blob extracted by ActivBlobFinder.

Features are calculated not only for objects, but also for each ROI, in form of the number of extracted objects, and the number of objects which have been evaluated as "okay" (so-called *good objects*) and "not okay" (*bad objects*), respectively.

Similarly, for each tool the number of objects summed over all ROIs of the tool are calculated, and the corresponding number of objects which have been evaluated as "okay" and "not okay", respectively. Further features are the number of ROIs which have been evaluated as "okay" (*good ROIs*) and "not okay" (*bad ROIs*), respectively.

Besides the value of each feature, its evaluation forms a separate result. Furthermore, the objects, ROIs, tools, and the overall application are evaluated.

This hierarchy of results is depicted in figure 1.1.

Evaluating by Formulating Conditions

ActivDecision lets you compare the value of an individual feature with two boundary values, a minimum and a maximum value. Thus, you may formulate conditions like *feature x must lie inside a and b*, i.e., $a \leq x \leq b$, or *feature x must be equal to a*, i.e., $x = a$. Of course, you may choose not to evaluate a feature at all; this is the default setting for all features.

A tool, ROI, or object is evaluated as "okay" if *all* of its features are evaluated as "okay", i.e., meet their corresponding conditions or do not have an attached condition. Similarly, the overall application is evaluated as "okay" if all tools are evaluated as "okay".

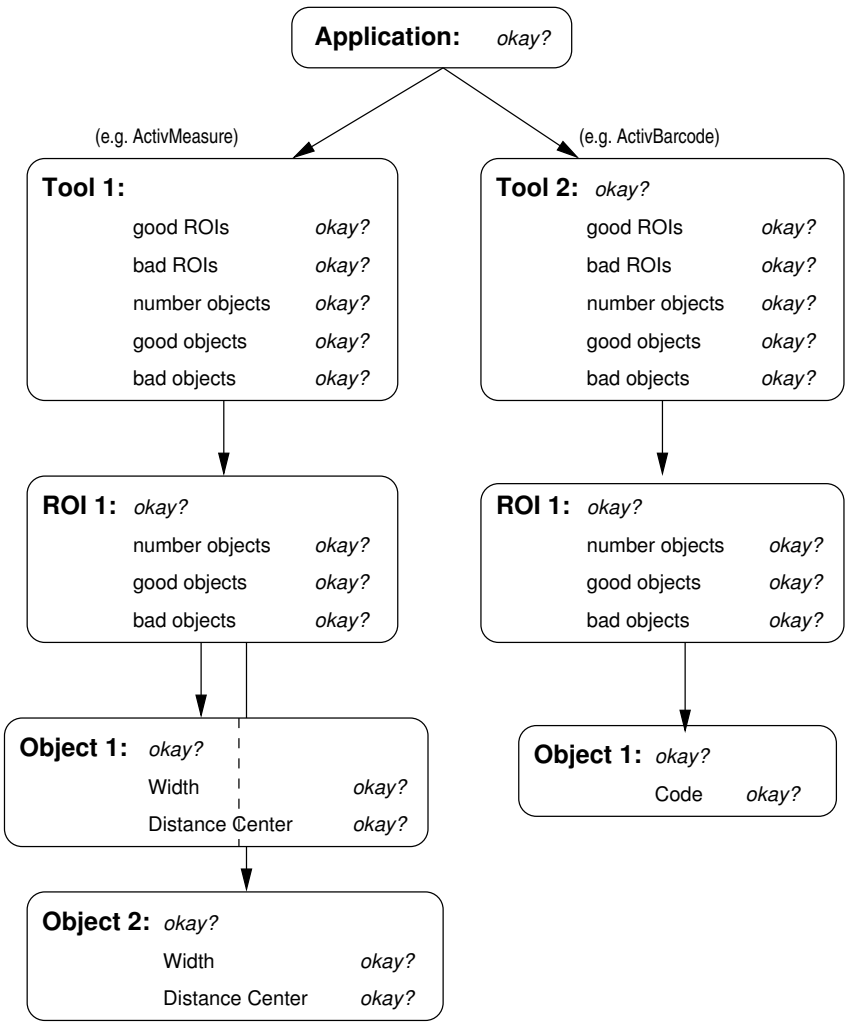


Figure 1.1: The hierarchy of features and evaluations in an example ActivVisionTools application.

Specifying Defaults for the Evaluation Parameters

In many applications, many similar objects are extracted which all should meet the same conditions. For this, ActivDecision allows to specify *default conditions* for individual features. These default conditions can be specified on different levels: *ROI-wide defaults* are valid for all objects

in a certain ROI, while *tool-wide defaults* are valid for all objects in all ROIs of a certain tool.

Of course, you can also mix default conditions with individual conditions: In such a case, an individual condition overrides a default condition. Similarly, an ROI-wide default overrides a tool-wide default. Note that you can override individual parameters of a condition separately while keeping default values for the rest.

1.2 The Sub-Tools of ActivDecision

ActivDecision provides three *master tools* and three *support tools*. In figure 1.2 they are shown in an example ActivVisionTools application, where they are used to evaluate the results of ActivMeasure.

`AVTDecision` is the main *master tool* of ActivDecision. It displays the overall evaluation of your application. How to use and configure `AVTDecision` is described in more detail in section 2.1 on page 8.

`AVTDecisionViewResults` is a *support tool* of ActivDecision. It allows to specify conditions for all types of features and displays the current feature values together with the evaluation results. How to use `AVTDecisionViewResults` is described in section 2.1 on page 8 and section 2.2 on page 10.

`AVTDecisionViewDefaults` is a *support tool* of ActivDecision. It allows to specify tool-wide and ROI-wide defaults for the conditions. How to use `AVTDecisionViewDefaults` is described in section 2.4 on page 14.

`AVTDecisionResult` is a *master tool* of ActivDecision, i.e., it can be used without `AVTDecision` being placed on the form. It allows to display selected feature values and or evaluations in different ways. How to use `AVTDecisionResult` is described in section 2.5 on page 16.

`AVTDecisionStatus` is a *master tool* of ActivDecision, i.e., it can be used without `AVTDecision` being placed on the form. It allows to display the overall status of the application in different ways. How to use `AVTDecisionStatus` is described in section 2.5 on page 16.

`AVTDecisionStatusParameters` is a *support tool* of `AVTDecisionStatus`. With it, you can configure the display of the overall status. How to use `AVTDecisionStatusParameters` is described in section 2.5 on page 16.

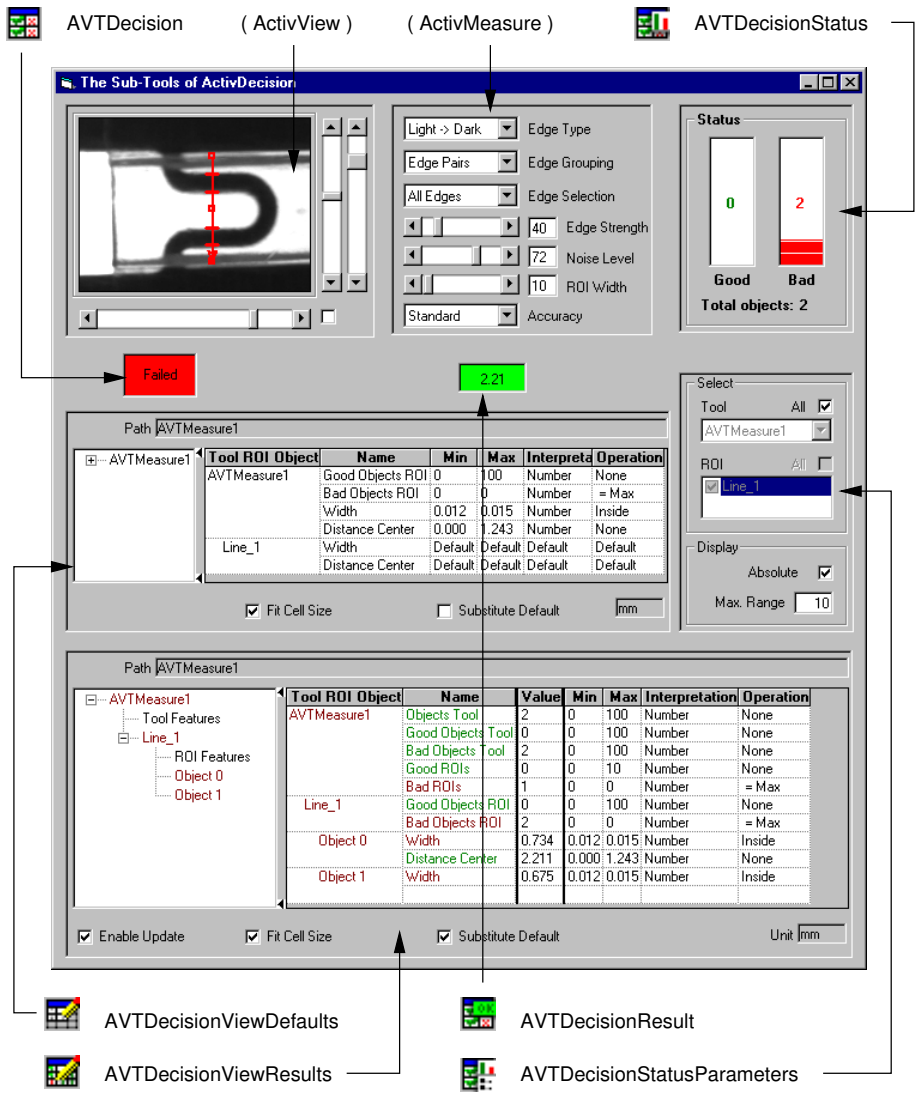


Figure 1.2: ActivDecision together with suitable other tools in a measuring application.

Chapter 2

Using ActivDecision

This chapter will explain how to use ActivDecision to evaluate the results of other ActivVision-Tools.

In the corresponding Visual Basic project (one for all sections), the task is to inspect car fuses using ActivMeasure and ActivAlignment. If you want to experiment with the example project in more detail we recommend to read the User's Manual for ActivMeasure and the User's Manual for ActivAlignment.

2.1 Browsing Results 8
2.2 Formulating Conditions 10
2.3 Specifying Defaults Via a Context Menu 12
2.4 Specifying Defaults Via the Support Tool 14
2.5 Displaying Selected Results 16

2.1 Browsing Results Using and

ActivDecision's *master tool* `AVTDecision` displays the overall evaluation of the application, while the *support tool* `AVTDecisionViewResults` allows both to view the results, i.e., the values of the tool, ROI, and object features (see section 1.1 on page 2) and their evaluation, and to formulate conditions the features have to fulfill. In this section, we concentrate on the former; how to formulate conditions is the topic of the next section. Section 2.5 on page 16 shows how to display selected results using `AVTDecisionResult` and `AVTDecisionStatus`.

Visual Basic Example

Preparation for the following example:

- ☐ Please open the project `using\decision_using.vbp` as the measuring application used in the following sections is already configured suitably in it.
- ☐ Execute the application (`Run` ▷ `Start` or via the corresponding button). Load the image `fuse\fuse25_03`.

The following steps are visualized in figure 2.1.

① Open `AVTDecisionViewResults` by clicking on `AVTDecision` with the right mouse button and selecting `Decision Results` in the popup menu.

② First, switch on the update of results and evaluations by checking ☑ `Enable Update`. We recommend to disable the update while experimenting with the parameters of other tools, especially ActivBlobFinder, to speed up the processing. Note that the check box only influences the display, the evaluations themselves are performed automatically as soon as you have formulated conditions.

③ The left part of `AVTDecisionViewResults` contains a navigation tree. At first, this tree is folded, showing only the connected tool(s). You can unfold and fold it by clicking on the ⊞ and ⊟ icons.

④ When you click on a label in the tree, the right part of `AVTDecisionViewResults` displays the corresponding features including their current values. The selected item is displayed in the top line. In figure 2.1, `AVTMeasure1` was selected, thus all the features of this tool, of its three ROIs, and of the therein extracted objects (i.e., edge pairs) are displayed. When you click on an object, it is highlighted in `AVTView`. Note that **objects start with the index 0**!

Results are evaluated and displayed at each image processing cycle, i.e., whenever you grab a new image, but also when you modify an ROI or change a parameter.

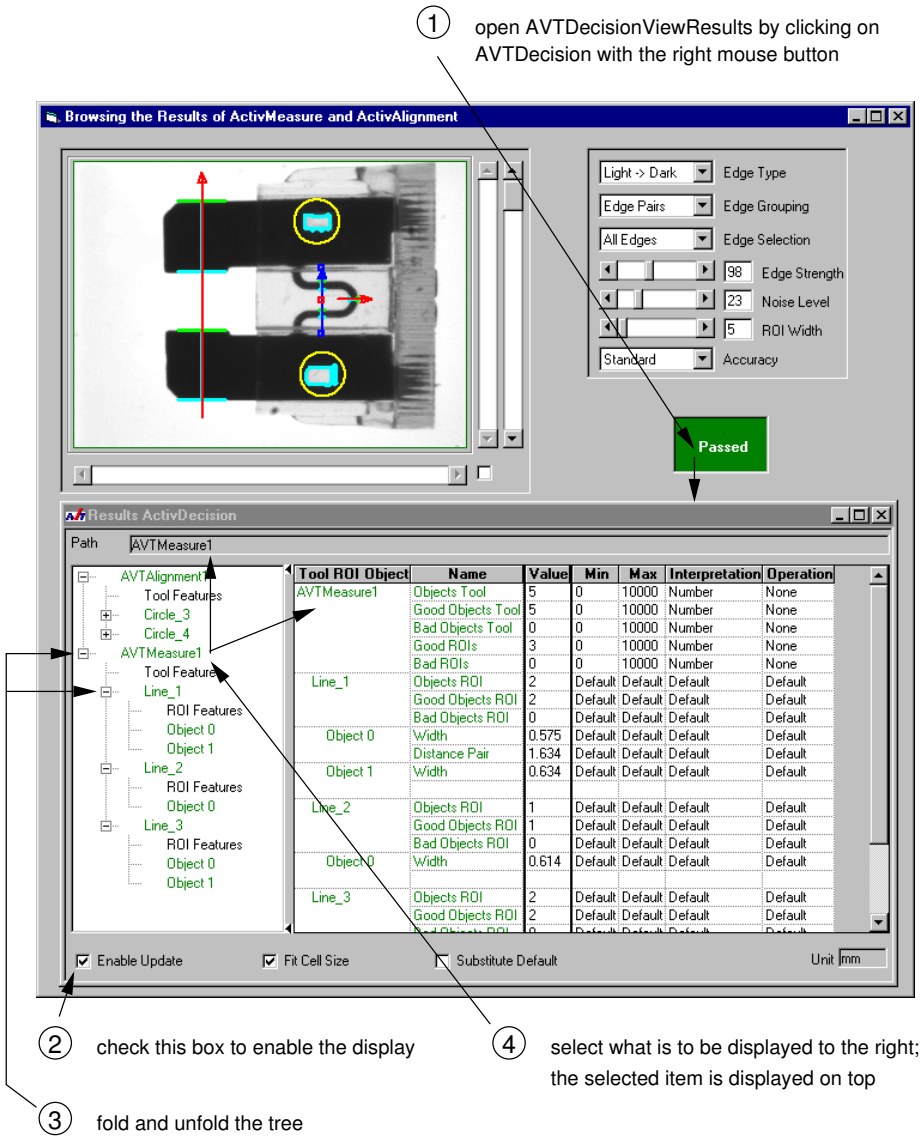


Figure 2.1: Browsing the results.

2.2 Formulating Conditions Using

`AVTDecisionViewResults` allows to formulate conditions for each individual feature on tool, ROI, and object level (see section 1.1 on page 2 for details on this hierarchy and on conditions themselves). How to specify default conditions is described in the following sections.

In our example application, car fuses are inspected using ActivMeasure and ActivAlignment as shown in figure 2.2: For the fuse to be evaluated as "okay", the width of the fuse wire, measured at three positions with two ROIs, should lie between `0.5` and `0.7mm`, and the distance between the two "fingers" between `4.2` and `4.8mm`. As the position and orientation of the inspected fuses can vary, the measurement ROIs are aligned to the fuse using the two rectangular gaps as the anchor.

Visual Basic Example

Preparation for the following example:

- ☐ If you did not already open it in the previous example, please open the project `using\decision_using.vbp` and execute it.
- ☐ Load the image sequence `fuse\fuse2.seq` and open `AVTDecisionViewResults` via a right mouse button click on `AVTDecision`.

The following steps are visualized in figure 2.2.

① In a condition, you can check a result value against a minimum value and/or a maximum value by entering values in the columns `Min` and `Max`. You can move between the fields with the cursor keys. Results can be interpreted as numbers or strings. You can change the interpretation in the column `Interpretation`. All features of ActivMeasure are interpreted as numbers, ActivAlignment's feature `Status` as a string.

② In the column `Operation` you can select various operations for the condition, e.g., that a feature value must lie `Inside` the two boundary values. If you select `None`, the feature is not evaluated; this is the default setting.

③ Whenever you move the cursor to another field or press `Enter`, all conditions are evaluated. In the column `Name`, those features which meet their condition appear in green, the others in red.

If one feature of an object does not meet its condition, the whole object is evaluated as "not okay" and the corresponding entry in the column `Tool ROI Object` and in the navigation tree is displayed in red. The colors and the text displayed in `AVTDecision` can be configured via `AVTViewProperties` (see User's Manual for ActivView), which can be opened by clicking on `AVTView` with the right mouse button and selecting `Properties` in the appearing context menu. In it, select the properties for `AVTDecision`.

④ The procedure for formulating conditions is the same for object, ROI, and tool features.

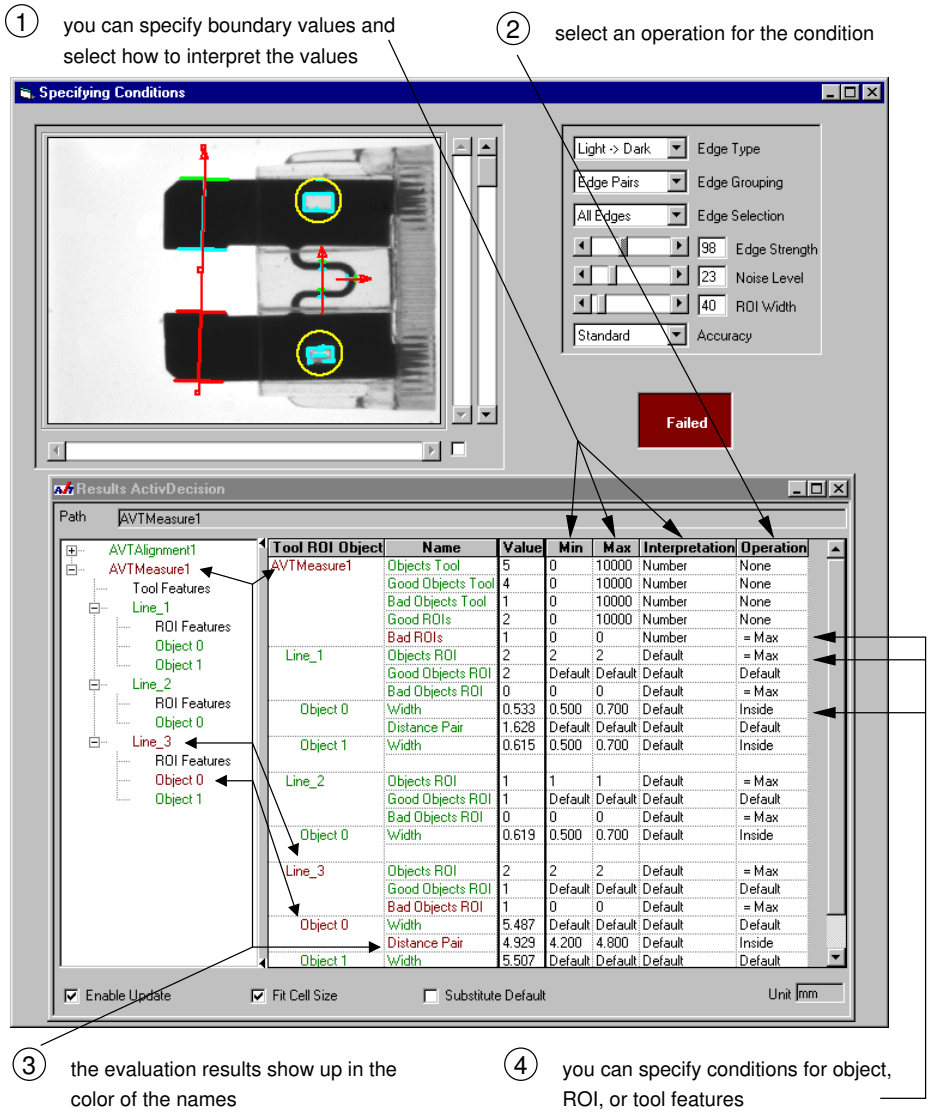


Figure 2.2: Formulating conditions for objects, ROIs, and tools.

Experiment with conditions and step through the image sequence to watch their effect. The sequence contains images of good and bad fuses; figure 2.2 shows suitable conditions.

2.3 Specifying Defaults Using

In some applications, many similar objects are extracted which all should meet the same conditions. To facilitate the formulation of conditions in such cases, ActivDecision allows to specify *default conditions* on the level of tools and ROIs (see section 1.1 on page 2).

Visual Basic Example

Preparation for the following example:

- ☐ If you did not already open it in the previous example(s), please open the project `using\decision_using.vbp` and execute it.
- ☐ Load the image sequence `fuse\fuse2.seq` and open `AVTDecisionViewResults` via a right mouse button click on `AVTDecision`.

The following steps are visualized in figure 2.3.

① If you click with the right mouse button on a field, the context menu `Display Defaults` allows to open a dialog to display and edit the corresponding default conditions. If, e.g., you click on the field `Min` of the `Width` of an object, the dialog contains tool-wide defaults for the object features as well as defaults for the corresponding ROI. If you click on the field `Max` of the `Bad Objects ROI` of an ROI the dialog contains the tool-wide defaults for the ROI features. Note that the dialog appears only once; to create the figure the second one was pasted on.

② In the dialog, you can formulate default conditions in the same way as described in section 2.2 on page 10.

③ An ROI-wide default condition overrides a tool-wide default. Furthermore, an individual condition for a feature overrides all default conditions. Figure 2.3 shows how to apply this mechanism to set suitable conditions for the example application described in section 2.2 on page 10. Experiment with default and individual conditions and step through the image sequence to watch the effect,

④ If you check ☑ `Substitute Default`, the entries `Default` are replaced by the actual values of the parameters.

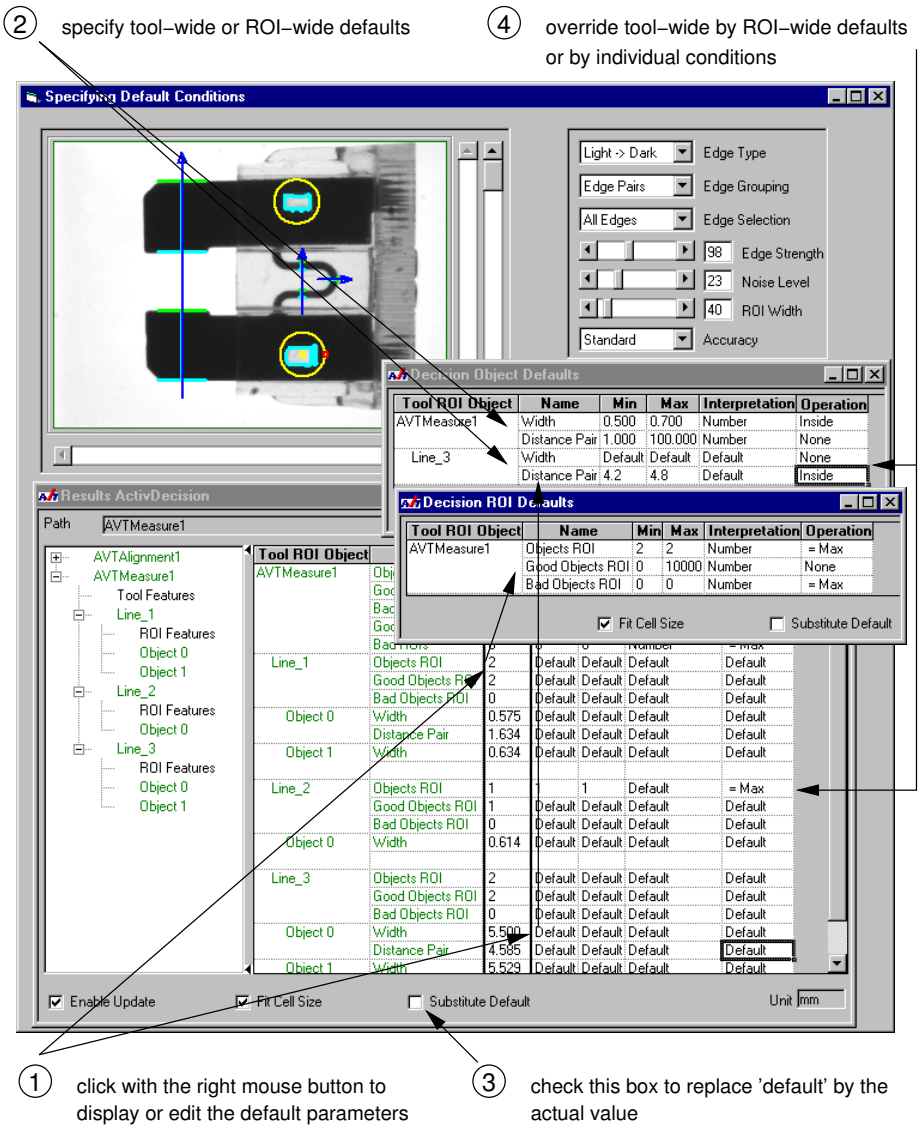


Figure 2.3: Specifying default conditions via the context dialog.

2.4 Specifying Defaults Using

In the previous section, default conditions were formulated via a context dialog of `AVTDecisionViewResults`. Alternatively, you can use the *support tool* `AVTDecisionViewDefaults`, which presents all default conditions in a hierarchical structure similar to `AVTDecisionViewResults`. Typically, the context dialog is used while developing an application, i.e., while designing suitable conditions. In contrast, if you already know which default conditions you want to set, `AVTDecisionViewDefaults` allows to specify them quickly all at once.

Visual Basic Example

Preparation for the following example:

- ☐ If you did not already open it in the previous example(s), please open the project `using\decision_using.vbp` and execute it.
- ☐ Load the image sequence `fuse\fuse2.seq`.

The following steps are visualized in figure 2.4.

① Open `AVTDecisionViewDefaults` by clicking on `AVTDecision` with the right mouse button and selecting `Decision Defaults` in the popup menu.

② Like `AVTDecisionViewResults`, `AVTDecisionViewDefaults` contains a navigation tree on the left side. When you click on a label in the tree, the right part of `AVTDecisionViewDefaults` displays the available default conditions. The selected item is displayed in the top line. In figure 2.4, `AVTMeasure1` was selected; thus, the right side contains the tool-wide default conditions for ROI and object features and the default conditions of its three ROIs for the object features.

③ On the right side, you can formulate default conditions as described in the previous section. An ROI-wide default condition overrides a tool-wide default. Furthermore, an individual condition for a feature overrides all default conditions. In figure 2.4, the default conditions are set like in figure 2.3 on page 13, which also shows the necessary individual conditions for the application.

④ If you check ☑ `Substitute Default`, the entries `Default` are replaced by the actual values of the parameters.

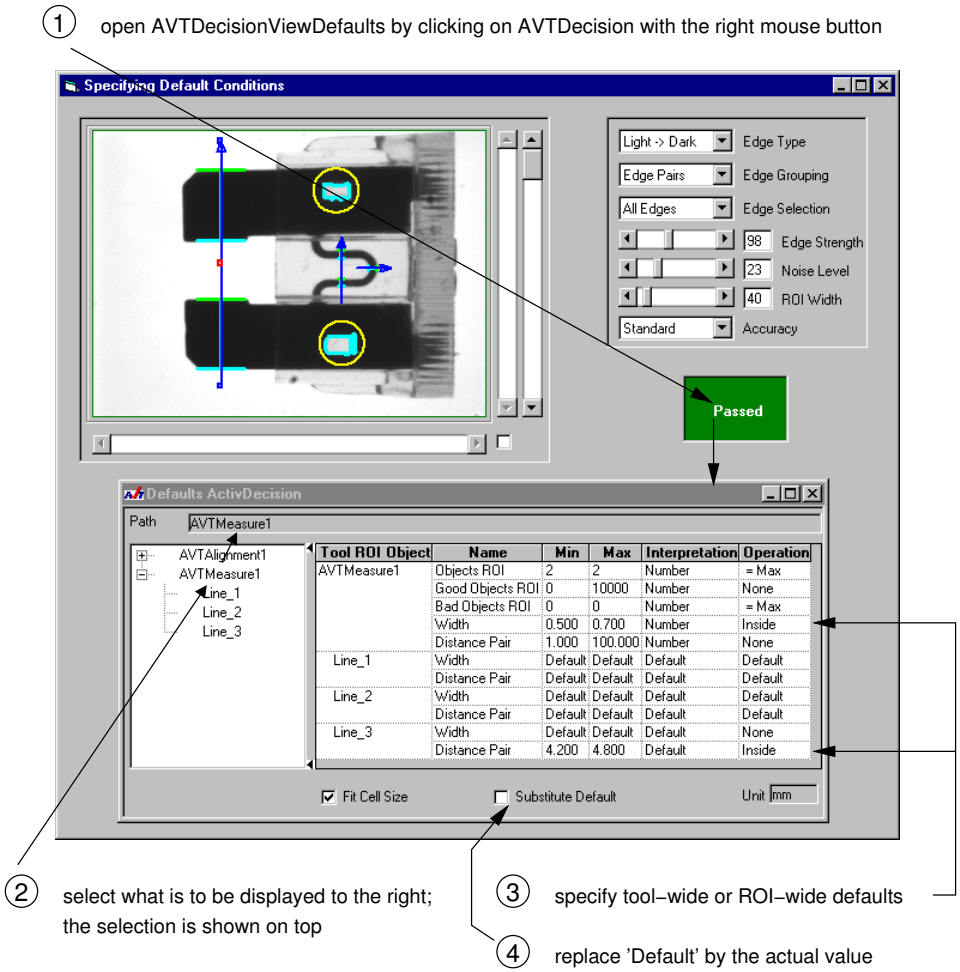


Figure 2.4: Specifying default conditions via `AVTDecisionViewDefaults` .

2.5 Displaying Selected Results Using and

Like ActivDataView, `AVTDecisionViewResults` displays all results at once. This is useful when setting up your machine vision application; in its final state, however, the application should typically display only selected results. For this, ActivDecision provides the two sub-tools `AVTDecisionResult` and `AVTDecisionStatus`. `AVTDecisionResult` allows to select an arbitrary feature, object, ROI, or tool and to display its value and/or its evaluation in various forms. `AVTDecisionStatus`, on the other hand, displays the number of "good" and "bad" objects of the overall application, of a tool, or of an ROI in form of bars, thus allowing to check the status of an application at a quick glance.

In fact, the two sub-tools are *master tools*, i.e., you can use them without placing `AVTDecision` on the form. However, without `AVTDecision` there are no evaluations, therefore the tools are typically used together with `AVTDecision`.

In this section, we describe but a few of the possible configurations of `AVTDecisionResult` and `AVTDecisionStatus`. We recommend to further experiment with these tools yourself. We start with `AVTDecisionStatus`. Please note that the example in figure 2.5 does not show a useful application of `AVTDecisionStatus`; a typical application would be to check the effect of changing processing parameters or evaluation boundaries if many objects are extracted.

Visual Basic Example

Preparation for the following example:

- ☐ If you did not already open it in the previous example(s), please open the project `using\decision_using.vbp`. At design time, add `AVTDecisionStatus` to the form by double-clicking the icon .
- ☐ Execute the application and load the image sequence `fuse\fuse2.seq`.

The following steps are visualized in figure 2.5.

① You can configure an instance of `AVTDecisionStatus` via `AVTDecisionStatus-Parameters`, which can be opened by clicking on `AVTDecisionStatus` with the right mouse button and selecting `Parameters` in the appearing context menu.

② In this dialog, you can select whose objects are to be analyzed via a set of check boxes in the frame `Select`. For example, you can choose to use the objects of all tools, of a specified tool, or of one or more ROIs of a tool.

③ Via the check box ☑ `Absolute` in the frame `Display`, you can specify whether absolut or relative numbers are to be displayed. If you choose absolute numbers, you can also specify the maximum of the displayed range.

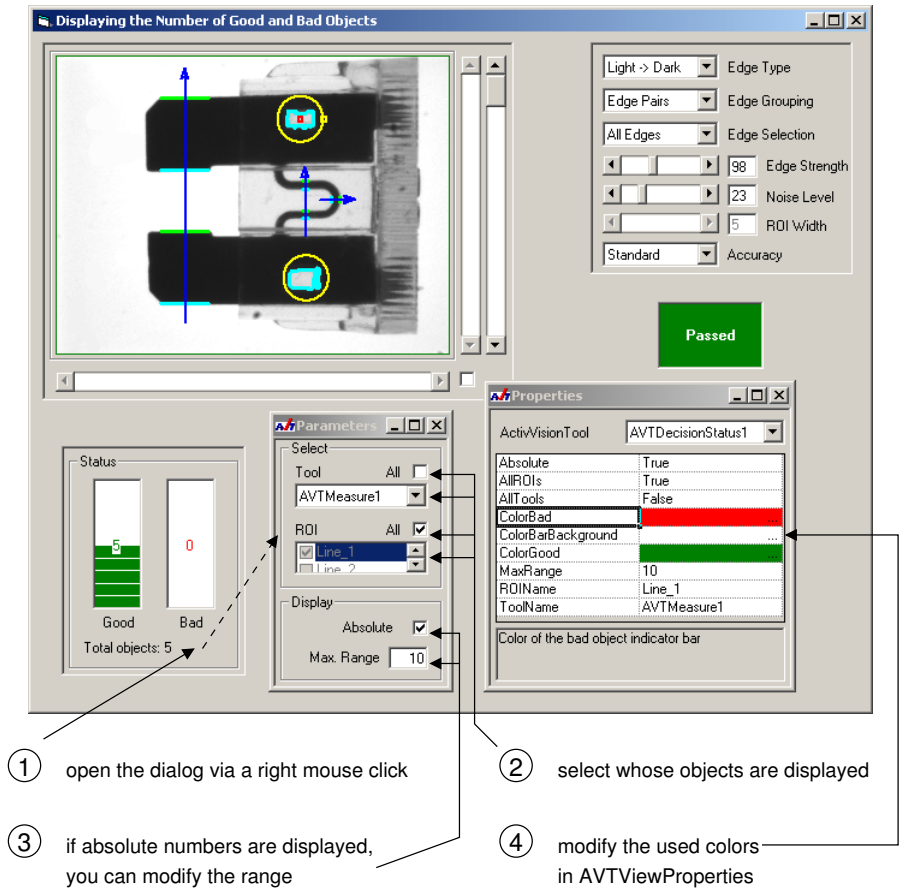


Figure 2.5: Displaying the number of good an bad objects in `AVTDecisionStatus` .

④ The colors used for the bars can be configured via `AVTViewProperties` (see User's Manual for ActivView, section 2.3 on page 20), which can be opened by clicking on `AVTView` with the right mouse button and selecting `Properties` in the appearing context menu. In it, select the properties for `AVTDecisionStatus`.

We continue with the example on the next double page.

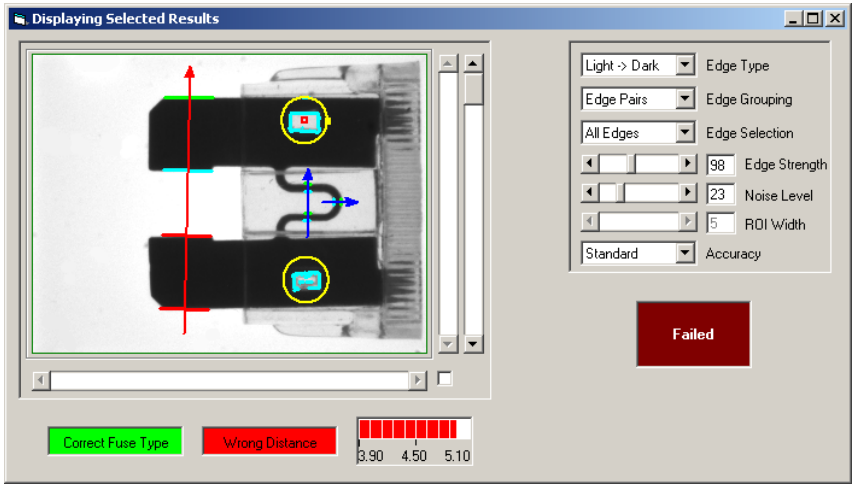


Figure 2.6: Examples for `AVTDecisionResult` .

(Displaying Selected Results, continued)

Now we turn to `AVTDecisionResult`. Figure 2.6 depicts three instances of this tool, which are configured to display the evaluation of the horizontal ROI over the fuse wire, the evaluation of the ROI over the fuse "fingers", and the value and evaluation of the distance between the fuse fingers.

Visual Basic Example

Preparation for the following example:

- ☐ At design time, add three instances of `AVTDecisionResult` to the form by double-clicking the icon and adapt their size. Execute the application and load the image sequence `fuse\fuse2.seq`.

You configure `AVTDecisionResult` partly at design time and partly at run time. At design time, you configure the appearance, e.g., whether the result is to be displayed in form of a bar or as plain text. At run time, you select the result and/or evaluation that is to be displayed; besides, you can also configure parts of the appearance, e.g., colors.

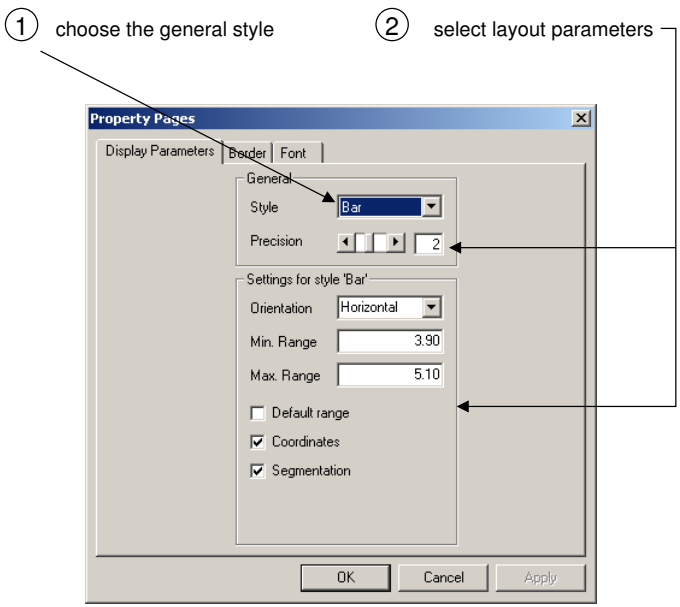


Figure 2.7: Configuring `AVTDecisionResult` at design time.

The following steps are depicted in figure 2.7.

1. Open the *property page* of `AVTDecisionResult` by clicking on it with the right mouse button and selecting `Properties` in the appearing context menu. In the combo box `Style`, you can select between three display styles.

2. Depending on the selected style, the lower part of the dialog shows different parameters. For the right-most `AVTDecisionResult` in figure 2.6, e.g., the style `'Bar'` was selected. For this style, you can, e.g., select the orientation of the bar, whether to use the default range, or else specify your own range.

   Please note that it depends on the programming environment whether all parameters can be modified and whether the selected values are saved in the DSC file. In Visual Basic 6.0, all parameters can be edited and are saved. Those parameters that cannot be edited in Visual Studio .NET or Visual C++ can be edited at run time as described on the next double page.

We continue with the example on the next double page.

(Displaying Selected Results, continued)

Now, we turn to the run-time parameters. For this, execute the application and load the image sequence `fuse\fuse2.seq`. The following steps are depicted in figure 2.8 (only parts of the application shown).

① You can configure an instance of `AVTDecisionResult` via two dialogs that can be opened by clicking on the instance with the right mouse button and selecting a dialog in the appearing context menu.

② If you select `Result Selection` in the context menu, a dialog appears in which you can select the result to display. Its left part contains the already well-known tree of results; to select an element for display, click on it with the left mouse button. At the top of the right part, the combo box `Result Type` allows to choose whether to display just the evaluation, just the result value, or both. In the example, we chose to display the value and evaluation of the distance between the fuse fingers.

③ When you select `Result Display Parameters` in the context menu, a dialog appears that contains the same elements as the property page of `AVTDecisionResult` shown in figure 2.7. Some elements can not be edited now but only at design time.

For the middle instance of `AVTDecisionResult` in figure 2.6, we chose to display just the evaluation of the ROI over the fingers in the style `'Plain'`. For this style, you can specify which text is to be displayed for the evaluations "good" and "bad", respectively. The text in the box `Error Caption` appears in case of an error, e.g., if you selected a feature that is not calculated in the current image. The cause of such an error is shown in the dialog `Result Selection` beside the label `Feature Data`; (not shown in the figure).

④ The used colors for the bars can be configured via `AVTViewProperties` (see User's Manual for ActivView, section 2.3 on page 20), which can be opened by clicking on `AVTView` with the right mouse button and selecting `Properties` in the appearing context menu. In it, select the properties for `AVTDecisionResult`.

In figure 2.6, we configured the left-most instance of `AVTDecisionResult` to display the evaluation of the horizontal ROI over the fuse wire with modified evaluation texts. Try to do this yourself and then step through the image sequence and observe the effect.

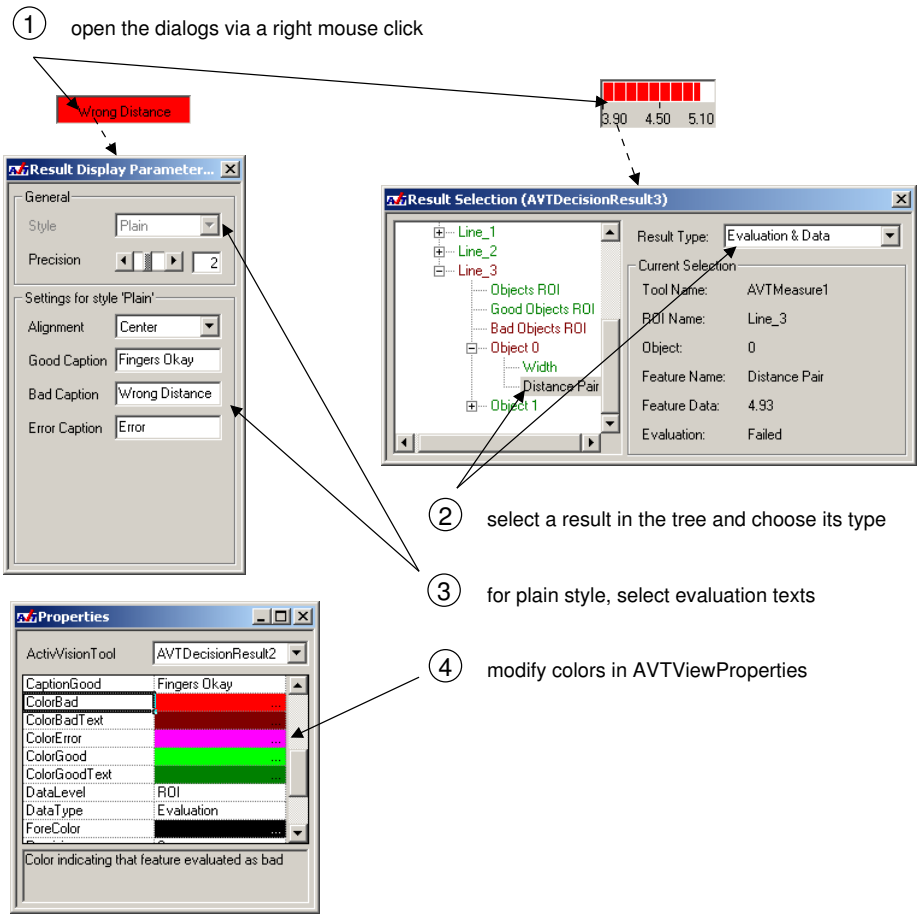


Figure 2.8: Configuring `AVTDecisionResult` at run time.

Chapter 3

Tips & Tricks

This chapter contains additional information that facilitates working with ActivDecision, e.g., how to modify the graphical display of results and to customize the appearance of an ActivVisionTools application in the two execution modes. Furthermore, it shows how to access evaluation results directly via the programming interface of ActivVisionTools.

3.1 Adapting the Display of Results in `AVTView` 24
3.2 Configuring the Two Execution Modes 26
3.3 Miscellaneous 28
3.4 Accessing Evaluation Results Via the Programming Interface 30
3.4.1 Basic Principles 31
3.4.2 An Example Project 34

3.1 Adapting the Display of Results Using

The previous chapter showed how the evaluation results are displayed in the sub-tools of ActivDecision. You can modify the colors marking “good” and “bad” items via properties.

Besides the display within ActivDecision, evaluation results also show up in `AVTView`, i.e., together with the image and the results of other tools. You can adapt this display using `AVTViewDisplayModes`, which is a *support tool* of ActivView.

Visual Basic Example

Preparation for the following example:

- ☐ If you worked on the example in the previous chapter, you may continue using this project.

  Otherwise, open the project `tips\decision_tips.vbp`, execute it, and load the image sequence `fuse\fuse2.seq`.
- ☐ Open `AVTDecisionViewResults` via a right mouse button click on `AVTDecision`.

The following steps are visualized in figure 3.1.

1. Open `AVTViewDisplayModes` by clicking on `AVTView` with the right mouse button and selecting `Display Modes` in the popup menu.
2. Objects and ROIs evaluated as “not okay” can be marked in `AVTView` with a special color. This color can be selected in the combo boxes `Error ROI` and `Error` of `AVTViewDisplayModes`.
3. The color in the combo box `Selected` is used to highlight objects when you click on them in the tree.
4. The colors marking the evaluation of the overall application in `AVTDecision` and the “good” and “bad” features, objects, ROIs, or tools in `AVTDecisionViewResults` can be set in `AVTViewProperties`, which can be opened by clicking on `AVTView` with the right mouse button and selecting `Properties` in the appearing context menu. In it, select the properties for `AVTDecision`.

   Likewise, the texts used in `AVTDecision` to indicate the overall evaluation are stored in the `AVTDecision`’s properties `TextGood` and `TextBad`.

If you step through the image sequence, you can watch the effect for those fuses that don’t meet the conditions.

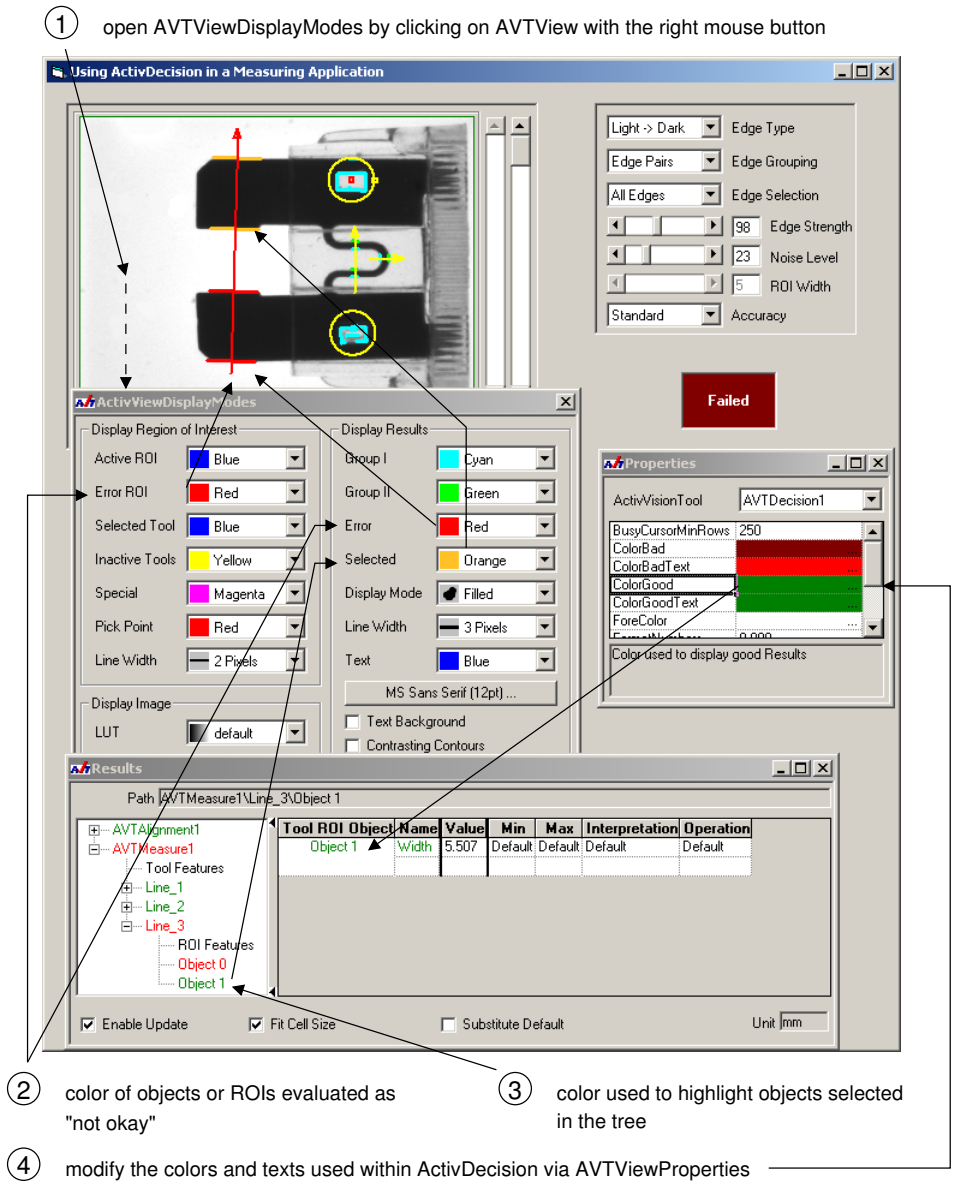


Figure 3.1: Modifying the display of evaluation results.

3.2 Configuring the Two Execution Modes Via and

In an ActivVisionTools application you can switch between two execution modes: the *configuration mode* and the *application mode*. The former should be used to setup and configure an application, the latter to run it. ActivView's *support tools* `AVTViewExecute` and `AVTViewConfigExec` allow you to switch between the two modes and to customize the behavior of an ActivVisionTools application in the two execution modes, e.g., display live images only in the *configuration mode* to setup your application, but then switch it off in the *application mode* to speed up the application.

Visual Basic Example

Preparation for the following example:

- ☐ If you worked on the previous example, you may continue using this project.

  Otherwise, open the project `tips\decision_tips.vbp`, execute it, and load the image sequence `fuse\fuse2.seq`.

The following steps are visualized in figure 3.2.

1. Open `AVTViewExecute` and `AVTViewConfigExec` by clicking on `AVTView` with the right mouse button and selecting `Execution` and `Execution Parameters` in the popup menu.
2. In `AVTViewExecute`, you can switch between the two modes via the combo box `Mode`.
3. To execute one cycle, press `Single`. In our example, this means that the next image of the sequence is loaded. With the other two buttons you can start the application in a continuous mode and stop it again.
4. For each of the two execution modes, you can choose what is to be displayed by checking the corresponding boxes in `AVTViewConfigExec`. Furthermore, you can specify if images can be dragged to the image window and whether ROIs can be modified in the two modes; by default, this is disabled in the *application mode* to prevent you from accidentially moving or deleting an ROI.

More information about these tool can be found in the User's Manual for ActivView, section 3.4 on page 34.

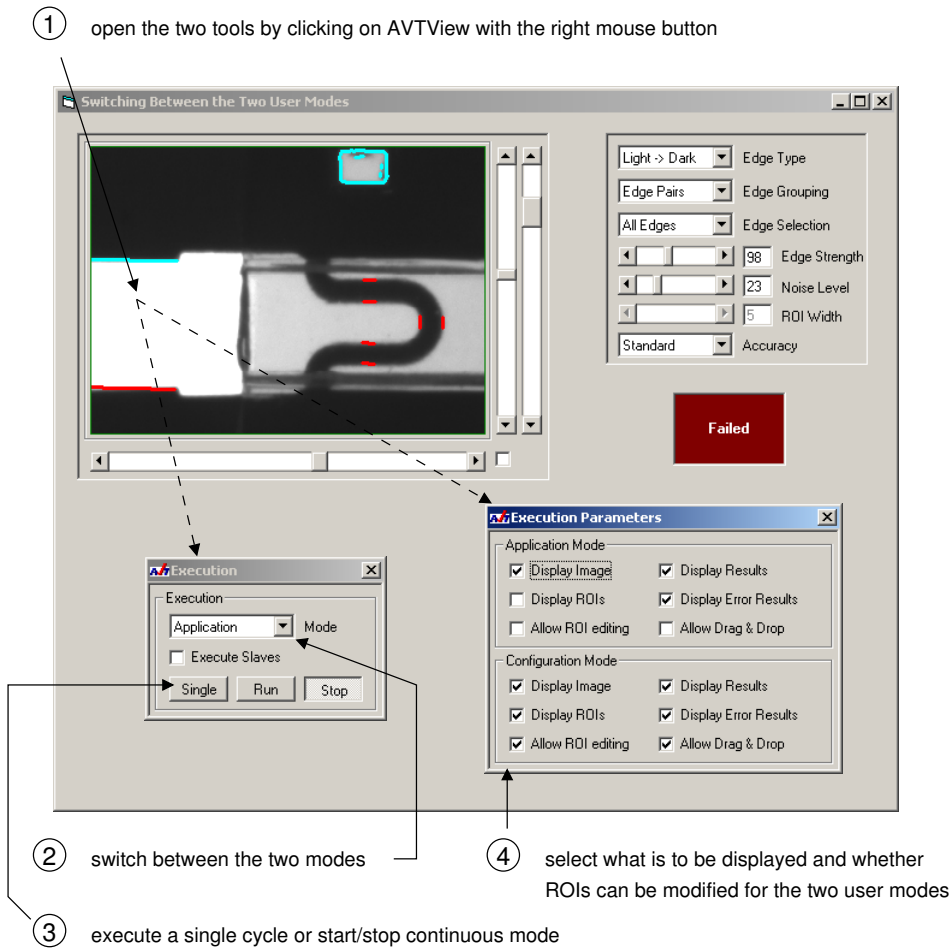


Figure 3.2: Customizing and switching between the two execution modes.

3.3 Miscellaneous

This section is a loose assortment of information about elements of ActivDecision which have not been described in the previous sections.

Visual Basic Example

Preparation for the following example:

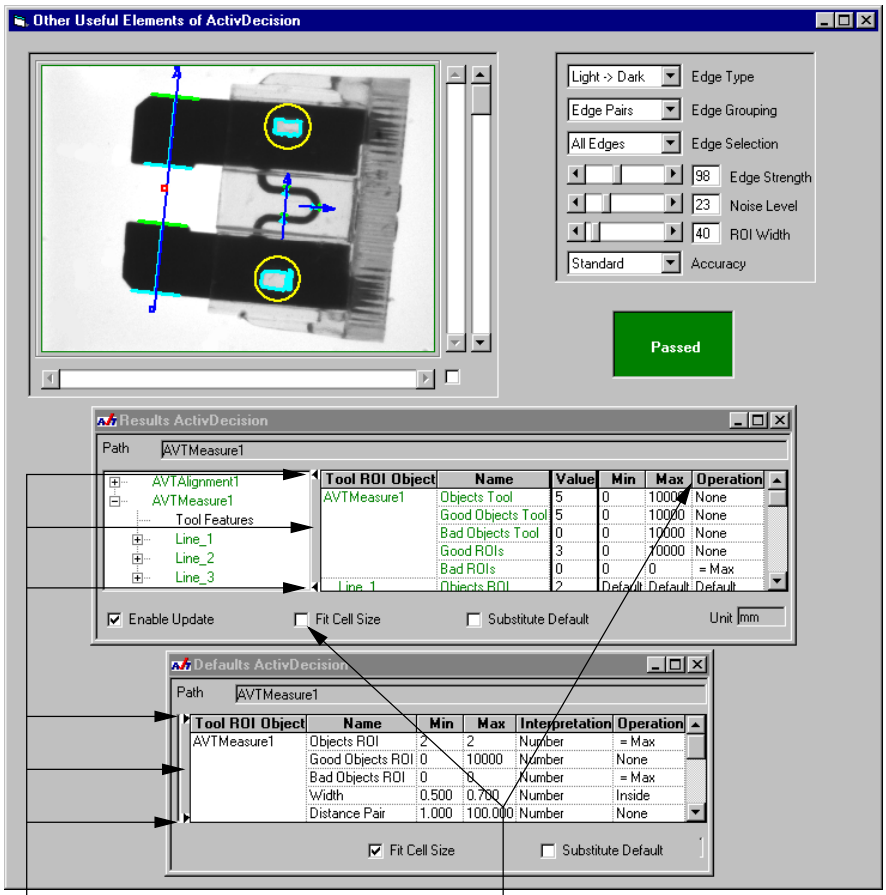
- ☐ If you worked on the previous example, you may continue using this project.

  Otherwise, open the project `tips\decision_tips.vbp`, execute it, and load the image sequence `fuse\fuse2.seq`.
- ☐ Open `AVTDecisionViewResults` and `AVTDecisionViewDefaults` via a right mouse button click on `AVTDecision`.

The following steps are visualized in figure 3.3.

① Both in `AVTDecisionViewResults` and `AVTDecisionViewDefaults` you can move the bar separating the tree from the data grid by dragging, i.e., by clicking on it with the left mouse button and then moving the mouse while keeping the button pressed. Furthermore, you can let the tree disappear by clicking on one of the arrow heads; in figure 3.3 this has been performed for `AVTDecisionViewDefaults`.

② If the box ☑ `Fit Cell Size` is checked, the fields in the data grid are resized automatically to fit the contained data. If this box is not checked, you can change the width of a column by dragging the corresponding separator in the head row. In figure 3.3, the column `Interpretation` of `AVTDecisionViewResults` has been "closed" by this method.



① move the separating bar or hide the tree

② if this box is not checked, you can change the size of fields

Figure 3.3: Miscellaneous Elements of ActivDecision .

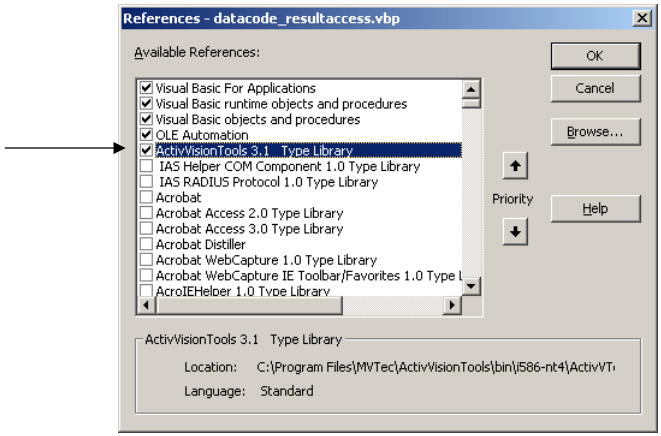
3.4 Accessing Evaluation Results Via the Programming Interface

The previous chapters and sections showed how to use ActivVisionTools interactively, i.e., via the graphical user interfaces presented by the underlying ActiveX controls. In this mode, you can develop the image processing part of your machine vision application rapidly and easily, without any programming. However, there is more to ActivVisionTools than the graphical user interfaces: Because ActivVisionTools comes as a set of ActiveX controls, it provides you with an open programming interface, thereby offering full flexibility.

In this section, we show how to access the evaluation results via the programming interface. With this, you can, e.g., realize an application-specific graphical user interface, perform additional processing on the results, or send results to a special output device. Detailed information about the programming interface can be found in the **Reference Manual**.

As in the previous sections, the examples stem from Visual Basic 6.0; if the (ActivVisionTools-specific) code differs in Visual Basic .NET, the corresponding lines are also shown (for the first appearance only). For other .NET languages or C++, please refer to the Advanced User's Guide for ActivVisionTools, section 1.2.3 on page 5 and section 1.3.4 on page 28, respectively. Please note that we assume that readers of this part have at least a basic knowledge of Visual Basic.

! To work with the programming interface, in VB 6.0 you must first **add the ActivVisionTools type library to the project's references** by checking the box labeled `ActivVisionTools Type Library` in the menu dialog `Project` ▷ `References`. In Visual Basic .NET, the reference is added automatically.



3.4.1 Basic Principles

The principal idea behind accessing the results of an ActivVisionTool is quite simple: When a tool has finished its execution, it raises an event called `Finish`, sending its results as a parameter. If you want to access the results, all you have to do, therefore, is to create a corresponding event procedure which handles the event.

Within the Visual Basic environment, you can create event procedures very easily as shown in figure 3.4: In the header of the form's code window there are two combo boxes. Select the instance of `AVTDecision` (by default called `AVTDecision1`) in the left combo box. The right combo box then lists all events available for this object; when you select `Finish`, the event procedure is created automatically.

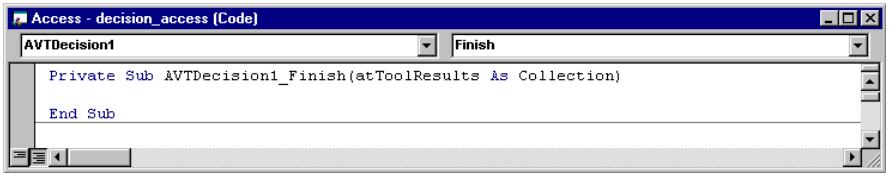


Figure 3.4: Creating a procedure to handle the event `Finish` .

Note that the event procedure returns the results of all those tools which are evaluated by ActivDecision in form of a `Collection` of `ACTIVVTOOLSLib.AVTToolResult`. These results contain both the feature values and the corresponding evaluations. In other words, ActivDecision itself produces no own results; the evaluations are attached to the result objects of the evaluated tools.

In Visual Basic .NET , the event handler has a different signature:

```
Private Sub AxAVTDecision1_Finish(ByVal sender As System.Object, _
                                  ByVal e As _
                                    AxActivVTools.__AVTDecision_FinishEvent) _
                                  Handles AxAVTDecision1.Finish
```

A first difference is that tool names start the prefix `Ax`, i.e., `AVTDecision` becomes `AxAVTDecision`. The main difference, however, is that the tool results are not directly passed; instead, they are encapsulated in the parameter `e`. From there, they can be extracted with the following lines; note the use of `VBA.Collection` instead of `Collection`!

```
Dim atToolResults As VBA.Collection
atToolResults = e.atToolResults
```

To use classes like `ACTIVVTOOLSLib.AVTToolResult` without the namespace `ACTIVVTOOLSLib` as in the code above, you must import this namespace by inserting the following line at

the very beginning of the code (more information about importing namespaces can be found in the Advanced User's Guide for ActivVisionTools in section 1.2.4.5 on page 12):

```
Imports ACTIVVTOOLSLib
```

The following code fragment searches the collection for the results of `AVTMeasure1` and "stores" them in the object variable `atMeasureResult`, or exits if no results are found:

```
Dim atToolResult As AVTToolResult
Dim atMeasureResult As AVTToolResult
Dim bMeasureResultsFound As Boolean

bMeasureResultsFound = False
For Each atToolResult In atToolResults
  If atToolResult.Name = "AVTMeasure1" Then
    Set atMeasureResult = atToolResult
    bMeasureResultsFound = True
  End If
Next
If bMeasureResultsFound = False Then
  Exit Sub
End If
```

As already remarked in the previous section, **in Visual Basic .NET** tool names are prefixed with `Ax`, thus you must search for the results of `AxAVTMeasure1`:

```
If atToolResult.name = "AxAVTMeasure1" Then
```

A tool result object contains the result data for all ROIs of your instance of `AVTMeasure`. Continuing the example code from above, the current number of ROIs of `AVTMeasure1` can be queried via

```
Dim iNumROIs As Integer

iNumROIs = atMeasureResult.ROINum
```

The results of a certain ROI can be accessed by specifying its name in a call to the method `ROIResult`, or by specifying its index in a call to the method `ROIResults`. The following code uses the latter method to access the first ROI of `AVTMeasure1`:

```
Dim atROIResult As AVTROIResult

Set atROIResult = atMeasureResult.ROIResults(0)
```

Now, we can, e.g., query the number of objects extracted in the ROI via

```
Dim iNumObjects As Integer

iNumObjects = atROIResult.ObjectNum
```

Actual measurement results for an object, i.e., the calculated values of features like `Distance` or `Width` can be accessed by specifying the ID of the object and the feature of interest in a call to the method `ObjectValue` of `ACTIVVTOOLSLib.AVTROIResult`. The feature handles are available as methods of the corresponding tool, e.g., `AVTMeasure.FeatureHandlePairWidth` being the handle for the measured width.

You can query the overall evaluations at different levels, tool, ROI, or object:

```
Dim bToolIsOK As Boolean, bROIIsOK As Boolean, bObjIsOK As Boolean

bToolIsOK = atMeasureResult.Evaluation
bROIIsOK = atROIResult.Evaluation
bObjIsOK = atROIResult.ObjEvaluation(0)
```

Furthermore, you can access the evaluation of individual features like the measured width of the object with the ID `0`, but also of tool features (e.g., the evaluation of the number of "bad" ROIs) or of ROI features (e.g., the evaluation of the number of objects) via the corresponding feature handle. In contrast to object features, the handles for tool and ROI features are available as properties of `ACTIVVTOOLSLib.AVTToolResult`.

```
Dim handleWidth As Integer, handleBadROIs As Integer
Dim handleObjectsInROI As Integer
Dim bWidthIsOK As Boolean, bBadROIsIsOK As Boolean
Dim bObjectsInROIIsOK As Boolean

handleWidth = AVTMeasure1.FeatureHandlePairWidth
handleBadROIs = atMeasureResult.FeatureHandleBadROIs
handleObjectsInROI = atMeasureResult.FeatureHandleObjectsROI

bWidthIsOK = atROIResult.ObjFeatureEvaluation(handleWidth, 0)
bBadROIsIsOK = atMeasureResult.FeatureEvaluation(handleBadROIs)
bObjectsInROIIsOK = atROIResult.FeatureEvaluation(handleObjectsInROI)
```

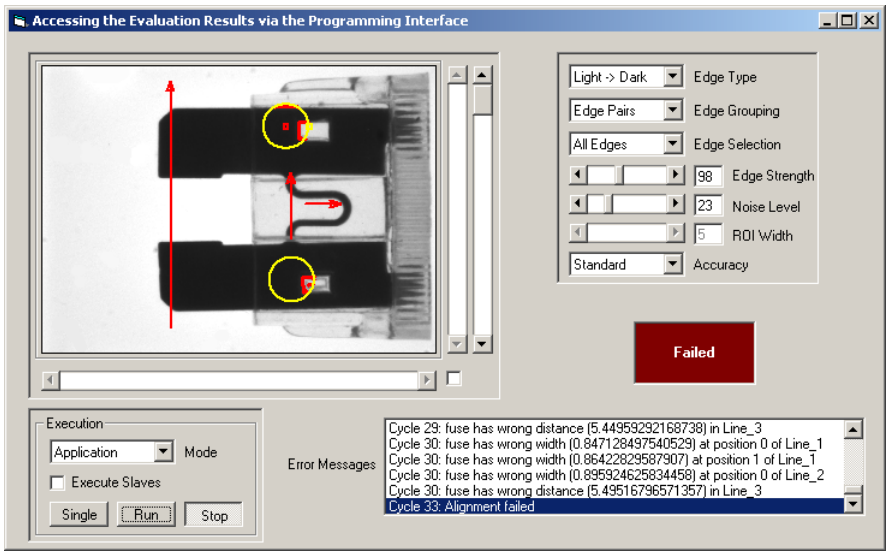


Figure 3.5: Accessing and displaying the evaluation results.

3.4.2 An Example Project

The ActivVisionTools distribution includes the example Visual Basic project `access\decision_access.vbp` which uses the methods described in the previous section to extend the application of chapter 2 on page 7. The example project has been configured in such a way, that the alignment fails for one image of the image sequence `fuse\fuse2.seq`; corresponding conditions check that the feature `Status` of `AVTAlignment1` is equal to `Success`. Furthermore, ActivAlignment's error mode has been set to `'Fail Feature'`, i.e., ActivMeasure does not perform any image processing if the alignment fails (see the User's Manual for ActivAlignment, section 3.1 on page 16, for more information).

When you start the example project and step through the image sequence, you can see that when the alignment fails, ActivMeasure and all of its ROIs are evaluated as "not okay" as well because the ROIs contain no objects. The task now is to check the evaluation of the two tools ActivMeasure and ActivAlignment via the programming interface and to display the "real" reason why the inspection has failed in a message list (see figure 3.5). The relevant project code is described in the following (only for Visual Basic 6.0!).

By placing the following code at the beginning of the handler of `AVTDecision`'s event `Finish`, the actual result access is restricted to the *application mode*. With this mechanism you can setup the vision part of your application in the configuration mode without having to worry about

run-time errors.

```
If Not AVTView1.ExecutionMode = eApplicationMode Then
  Exit Sub
End If
```

The first task is to extract the results of ActivMeasure and ActivAlignment; the following code solves this task including checks that the results are actually present:

```
Dim atToolResult As AVTToolResult
Dim bFoundMeasureResults As Boolean, bFoundAlignmentResults As Boolean
Dim atMeasureResult As AVTToolResult
Dim atAlignmentResult As AVTToolResult

bFoundMeasureResults = False
bFoundAlignmentResults = False
For Each atToolResult In atToolResults
  If atToolResult.Name = "AVTMeasure1" Then
    Set atMeasureResult = atToolResult
    bFoundMeasureResults = True
  Else
    If atToolResult.Name = "AVTAlignment1" Then
      Set atAlignmentResult = atToolResult
      bFoundAlignmentResults = True
    End If
  End If
Next
If (bFoundMeasureResults = False) Or (bFoundAlignmentResults = False) Then
  Exit Sub
End If
```

Then, the program checks whether the application contains three lines of measurement:

```
Dim iNumROI As Integer

iNumROI = atMeasureResult.ROINum
If Not iNumROI = 3 Then
  errorMsg = "Cycle " & iNumCycles & ": Corrupt application!"
  Call DisplayMessage(errorMsg)
  Call SetAlarm
  Exit Sub
End If
```

If not exactly three ROIs exist, the function `SetAlarm` stops the application and an error message is displayed. The function `ClearAlarm` resets the alarm. Test this behavior by creating an additional ROI.

```
Private bIsError As Boolean

Private Function SetAlarm()
  AVTView1.RunState = False
  bIsError = True
End Function

Private Function ClearAlarm()
  bIsError = False
End Function
```

If exactly three lines of measurement are present, their results are stored in an array:

```
Dim atROIResult(2) As AVTROIResult

For i = 0 To 2
  Set atROIResult(i) = atMeasureResult.ROIResults(i)
Next
```

Now, the evaluation of ActivAlignment is checked; if the alignment fails, an error message is displayed.

```
If Not atAlignmentResult.Evaluation = True Then
  ' first check whether this error is "new"
  If bIsNewCycle = True Then
    bIsNewCycle = False
    errorMsg = "Cycle " & iNumCycles & ": Alignment failed"
    Call DisplayMessage(errorMsg)
    Call SetAlarm
    Exit Sub
  Else
    Exit Sub
  End If
End If
```

One has to keep in mind that `AVTDecision` is executed not only when the next image is grabbed but also whenever you modify an ROI or a parameter. To distinguish the two cases an event raised by `AVTView` at the start of each execution cycle can be used to set a variable called `bIsNewCycle`:

```
Private bIsNewCycle As Boolean

Private Sub AVTView1_CycleStart()
  bIsNewCycle = True
End Sub
```

Before reacting on an evaluation in the event procedure this variable is checked (and imme-

diately reset). You can test this behavior by modifying a parameter of AVTMeasure after the alignment has failed: No new error message is displayed.

The error message also contains the number of the cycle in which the error occurred. The corresponding counter is incremented in the handler for AVTView's event CycleStart which was introduced already in the previous section:

```
Private iNumCycles As Integer

Private Sub AVTView1_CycleStart()
  iNumCycles = iNumCycles + 1
End Sub
```

If the alignment was successful, the evaluation of ActivMeasure is checked. If it is "not okay", the ROI evaluations are checked in their turn. The following code analyzes the cause of error further and displays corresponding error messages. If you let the application run continuously by clicking Run in AVTViewExecute, it will stop at other images with inspection errors.

```
For i = 0 To iNumROI - 1
  If atROIResult(i).Evaluation = False Then
    If atROIResult(i).FeatureEvaluation(handleObjectsInROI) = False Then
      errorMsg = "Cycle " & iNumCycles & _
                 ": wrong number of objects in " & sROINames(i)
      Call DisplayMessage(errorMsg)
    End If
    If atROIResult(i).FeatureEvaluation(handleBadObjectsInROI) = False Then
       For Each badObject In atROIResult(i).BadObjectIndices
         If i < 2 Then
           errorMsg = "Cycle " & iNumCycles & _
                      ": fuse has wrong width (" & _
                      atROIResult(i).ObjectValue(handleWidth, badObject) _
                      & ") at position " & badObject & " of " & _
                      sROINames(i)
           Call DisplayMessage(errorMsg)
         Else
           errorMsg = "Cycle " & iNumCycles & _
                      ": fuse has wrong distance (" & _
                      atROIResult(i).ObjectValue(handleWidth, badObject) _
                      & ") in " & sROINames(i)
           Call DisplayMessage(errorMsg)
        End If
      Next
    End If
  End If
Next
```

When using the programming interface of ActivVisionTools, you leave the safe world of the

graphical user interfaces where all input is checked for validity automatically. In contrast, if you try to access a non-existant object or result via the programming interface, a run-time error is caused which halts your application. To avoid this, you can use the Visual Basic error handling mechanisms, i.e., set up an error handler which examines any occurring error and reacts in a suitable way. In the example project, if an error is caused by the result access a popup with the error description appears and the function `SetAlarm` is called. To view the effect of the error handler, de-select the feature `Width` in `AVTMeasureResults`.

```
Private Sub AVTDecision1_Finish(atToolResults As Collection)

  ' variable declarations

On Error GoTo ErrorHandler

  ' procedure body

  Exit Sub

ErrorHandler:
  Dim sTitle As String
  If Left(Err.Source, 11) = "ActivVTools" Then
    sTitle = "ActivVisionTools Error"
  Else
    sTitle = "Runtime Error " & CStr(Err.Number)
  End If
  Call MsgBox(Err.Description, vbExclamation, sTitle)
  Call SetAlarm
End Sub
```